CSE 311 Foundations of Computing I

Autumn 2011
Lecture 29
Course Summary

Announcements

- Review sessions
  - Saturday, Dec 10, 4 pm, EEB 037 (Anderson)
  - Sunday, Dec 11, 4 pm, EEB 037 (Beame)
- Answer Catalyst Survey about which time you will take the final exam (by Sunday).
  - Review session Saturday/Sunday
  - List of Final Exam Topics and sampling of some typical kinds of exam questions on the web
- Final exam
  - Monday, Dec 12, 2:30-4:20 pm, Gug 220
  - Monday, Dec 12, 4:30-6:20 pm, Gug 220

About the course

- From the CSE catalog:
  - **CSE 311 Foundations of Computing I (4)** Examines fundamentals of logic, set theory, induction, and algebraic structures with applications to computing; finite state machines; and limits of computability. Prerequisite: CSE 143; either MATH 126 or MATH 136.
- What this course is about:
  - Foundational structures for the practice of computer science and engineering

Propositional Logic

- Statements with truth values
  - The Washington State flag is red
  - It snowed in Whistler, BC on January 4, 2011.
  - Rick Perry won the Iowa straw poll
  - Space aliens landed in Roswell, New Mexico
  - If n is an integer greater than two, then the equation $a^n + b^n = c^n$ has no solutions in non-zero integers a, b, and c.
- Propositional variables: *p, q, r, s, . . .*
- Truth values: **T** for true, **F** for false
- Compound propositions

Negation (not)	$\neg p$
Conjunction (and)	$p \wedge q$
Disjunction (or)	$p \vee q$
Exclusive or	$p \oplus q$
Implication	$p \rightarrow q$
Biconditional	$p \leftrightarrow q$

English and Logic

- You cannot ride the roller coaster if you are under 4 feet tall unless you are older than 16 years old
  - *q*: you can ride the roller coaster
  - *r*: you are under 4 feet tall
  - *s*: you are older than 16

$$\boxed{(r \wedge \neg s) \rightarrow \neg q}$$

Logical equivalence

- Terminology: A compound proposition is a
  - *Tautology* if it is always true
  - *Contradiction* if it is always false
  - *Contingency* if it can be either true or false

$p \vee \neg p$

$p \oplus p$

$(p \rightarrow q) \wedge p$

$(p \wedge q) \vee (p \wedge \neg q) \vee (\neg p \wedge q) \vee (\neg p \wedge \neg q)$

Logical Equivalence

- *p* and *q* are *logically equivalent* iff $p \leftrightarrow q$ is a tautology
- The notation $p \equiv q$ denotes *p* and *q* are logically equivalent

- De Morgan's Laws:

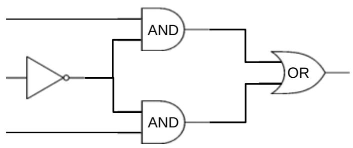
$$\neg (p \wedge q) \equiv \neg p \vee \neg q$$
$$\neg (p \vee q) \equiv \neg p \wedge \neg q$$

Digital Circuits

- Computing with logic
  - **T** corresponds to 1 or "high" voltage
  - **F** corresponds to 0 or "low" voltage

- Gates
  - Take inputs and produce outputs
    - Functions
  - Several kinds of gates
  - Correspond to propositional connectives
    - Only symmetric ones (order of inputs irrelevant)

Combinational Logic Circuits



Wires can send one value to multiple gates

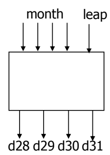
A quick combinational logic example

- Calendar subsystem: number of days in a month (to control watch display)
  - used in controlling the display of a wrist-watch LCD screen

  - inputs: month, leap year flag
  - outputs: number of days

Implementation as a combinational digital system

- Encoding:
  - how many bits for each input/output?
  - binary number for month
  - four wires for 28, 29, 30, and 31



month	leap	d28	d29	d30	d31
0000	–	–	–	–	–
0001	–	0	0	0	1
0010	0	1	0	0	0
0010	1	0	1	0	0
0011	–	0	0	0	1
0100	–	0	0	1	0
0101	–	0	0	0	1
0110	–	0	0	1	0
0111	–	0	0	0	1
1000	–	0	0	0	1
1001	–	0	0	1	0
1010	–	0	0	0	1
1011	–	0	0	1	0
1100	–	0	0	0	1
1101	–	–	–	–	–
1110	–	–	–	–	–
1111	–	–	–	–	–

Combinational example (cont'd)

- Truth-table to logic to switches to gates
  - d28 = "1 when month=0010 and leap=0"
  - d28 = m8'•m4'•m2•m1'•leap'

  - d31 = "1 when month=0001 or month=0011 or ... month=1100"
  - d31 = (m8'•m4'•m2'•m1) + (m8'•m4'•m2•m1) + ... (m8•m4•m2'•m1')
  - d31 = can we simplify more?

month	leap	d28	d29	d30	d31
0000	–	–	–	–	–
0001	–	0	0	0	1
0010	0	1	0	0	0
0010	1	0	1	0	0
0011	–	0	0	0	1
0100	–	0	0	1	0
...					
1100	–	0	0	0	1
1101	–	–	–	–	–
111–	–	–	–	–	–

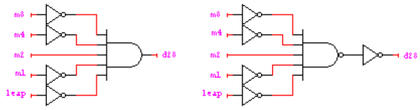
Combinational example (cont'd)

d28 = m8'•m4'•m2•m1'•leap'

d29 = m8'•m4'•m2•m1'•leap

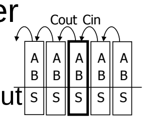
d30 = (m8'•m4•m2'•m1') + (m8'•m4•m2•m1') +
(m8•m4'•m2'•m1) + (m8•m4'•m2•m1)
= (m8'•m4•m1') + (m8•m4'•m1)

d31 = (m8'•m4'•m2'•m1) + (m8'•m4'•m2•m1) +
(m8'•m4•m2'•m1) + (m8'•m4•m2•m1) +
(m8•m4'•m2'•m1') + (m8•m4'•m2•m1') +
(m8•m4•m2'•m1')



A simple example: 1-bit binary adder

- Inputs: A, B, Carry-in
- Outputs: Sum, Carry-out



A	B	Cin	Cout	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Cout = B Cin + A Cin + A B

S = A′ B′ Cin + A′ B Cin′ + A B′ Cin′ + A B Cin
= A′ (B′ Cin + B Cin′) + A (B′ Cin′ + B Cin)
= A′ Z + A Z′
= A xor Z = A xor (B xor Cin)

Boolean algebra

George Boole – 1854

- An algebraic structure consists of
  - a set of elements B
  - binary operations { + , • }
  - and a unary operation { ' }
  - such that the following axioms hold:

1. the set B contains at least two elements: a, b

2. closure:	a + b is in B	a • b is in B
3. commutativity:	a + b = b + a	a • b = b • a
4. associativity:	a + (b + c) = (a + b) + c	a • (b • c) = (a • b) • c
5. identity:	a + 0 = a	a • 1 = a
6. distributivity:	a + (b • c) = (a + b) • (a + c)	a • (b + c) = (a • b) + (a • c)
7. complementarity:	a + a' = 1	a • a' = 0

Mapping truth tables to logic gates

- Given a truth table:
  1. Write the Boolean expression
  2. Minimize the Boolean expression
  3. Draw as gates
  4. Map to available gates

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

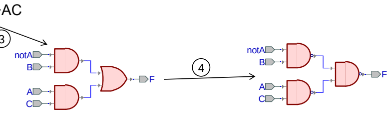
1

F = A'BC'+A'BC+AB'C+ABC

2

= A'B(C'+C)+AC(B'+B)
= A'B+AC

3



Sum-of-products canonical forms

- Also known as disjunctive normal form
- Also known as minterm expansion

F = *001* *011* *101* *110* *111*

F = A'B'C + A'BC + AB'C + ABC' + ABC

A	B	C	F	F'
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	0

F′ = A'B'C′ + A'BC′ + AB'C′

Predicate Calculus

- *Predicate* or *Propositional Function*
  - A function that returns a truth value
- "*x* is a cat"
- "student *x* has taken course *y*"
- "*x* > *y*"
- ∀ *x P(x)* : *P(x)* is true for every *x* in the domain
- ∃ *x P(x)* : There is an *x* in the domain for which *P(x)* is true

Statements with quantifiers

Domain: Positive Integers

Even(x)
Odd(x)
Prime(x)
Greater(x,y)
Equal(x,y)

- $\forall x$ (Even(x) $\vee$ Odd(x))
- $\exists x$ (Even(x) $\wedge$ Prime(x))
- $\forall x \exists y$ (Greater(y, x) $\wedge$ Prime(y))
- $\forall x$ (Prime(x) $\rightarrow$ (Equal(x, 2) $\vee$ Odd(x))
- $\exists x \exists y$(Equal(x, y + 2) $\wedge$ Prime(x) $\wedge$ Prime(y))

Proofs

- Start with hypotheses and facts
- Use rules of inference to extend set of facts
- Result is proved when it is included in the set

Simple Propositional Inference Rules

- Excluded middle $\dfrac{}{\therefore\ p \vee \neg p}$
- Two inference rules per binary connective one to eliminate it, one to introduce it.

$$\frac{p \wedge q}{\therefore\ p,\ q} \qquad \frac{p,\ q}{\therefore\ p \wedge q}$$

$$\frac{p \vee q,\ \neg p}{\therefore\ q} \qquad \frac{p}{\therefore\ p \vee q,\ q \vee p}$$

$$\frac{p,\ p \rightarrow q}{\therefore\ q} \qquad \frac{p \Rightarrow q}{\therefore\ p \rightarrow q}$$

Direct Proof Rul

Inference Rules for Quantifiers

$$\frac{P(c) \text{ for some } c}{\therefore\ \exists x\ P(x)} \qquad \frac{\forall x\ P(x)}{\therefore\ P(a) \text{ for any } a}$$

$$\frac{\text{“Let a be anything”...}P(a)}{\therefore\ \forall x\ P(x)} \qquad \frac{\exists x\ P(x)}{\therefore\ P(c) \text{ for some special } c}$$

Even and Odd

Even(x) $\equiv \exists y$ (x=2y)
Odd(x) $\equiv \exists y$ (x=2y+1)
Domain: Integers

- Prove: “The square of every odd number is odd”
  English proof of: $\forall x$ (Odd(x)$\rightarrow$Odd(x^2))

Let x be an odd number.
Then x=2k+1 for some integer k (depending on x)
Therefore $x^2=(2k+1)^2= 4k^2+4k+1=2(2k^2+2k)+1$.
Since $2k^2+2k$ is an integer, x^2 is odd. □

Characteristic vectors

- Let U = {1, . . ., 10}, represent the set {1,3,4,8,9} with

1011000110

- Bit operations:
  - 0110110100 $\vee$ 0011010110 = 0111110110
- **`ls –l`**

```
drwxr-xr-x ... Documents/
-rw-r--r-- ... file1
```

One-time pad

- Alice and Bob privately share random n-bit vector K
  - Eve does not know K
- Later, Alice has n-bit message m to send to Bob
  - Alice computes $C = m \oplus K$
  - Alice sends C to Bob
  - Bob computes $m = C \oplus K$ which is $(m \oplus K) \oplus K$
- Eve cannot figure out m from C unless she can guess K

Arithmetic mod 7

- $a +_7 b = (a + b) \bmod 7$
- $a \times_7 b = (a \times b) \bmod 7$

+	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	2	3	4	5	6	0
2	2	3	4	5	6	0	1
3	3	4	5	6	0	1	2
4	4	5	6	0	1	2	3
5	5	6	0	1	2	3	4
6	6	0	1	2	3	4	5

x	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6
2	0	2	4	6	1	3	5
3	0	3	6	2	5	1	4
4	0	4	1	5	2	6	3
5	0	5	3	1	6	4	2
6	0	6	5	4	3	2	1

Division Theorem

Let *a* be an integer and *d* a positive integer. Then there are *unique* integers *q* and *r*, with $0 \le r < d$, such that $a = dq + r$.

q = *a* **div** *d*   *r* = *a* **mod** *d*

Modular Arithmetic

Let a and b be integers, and m be a positive integer. We say a *is congruent to b modulo m* if m divides a – b. We use the notation $a \equiv b \pmod{m}$ to indicate that a is congruent to b modulo m.

Let a and b be integers, and let m be a positive integer. Then $a \equiv b \pmod{m}$ if and only if a mod m = b mod m.

Let m be a positive integer. If $a \equiv b \pmod{m}$ and $c \equiv d \pmod{m}$, then
$a + c \equiv b + d \pmod{m}$ and
$ac \equiv bd \pmod{m}$

Let a and b be integers, and let m be a positive integer. Then $a \equiv b \pmod{m}$ if and only if a mod m = b mod m.

Integer representation

Signed integer representation

Suppose $-2^{n-1} < x < 2^{n-1}$
First bit as the sign, n-1 bits for the value

99: 0110 0011, -18: 1001 0010

Two's complement representation

Suppose $0 \le x < 2^{n-1}$,
x is represented by the binary representation of x
-x is represented by the binary representation of 2^n-x

99: 0110 0011, -18: 1110 1110

Hashing

- Map values from a large domain, 0…M-1 in a much smaller domain, 0…n-1
- Index lookup
- Test for equality
- Hash(x) = x mod p
  - (or Hash(x) = (ax + b) mod p
- Often want the hash function to depend on all of the bits of the data
  - Collision management

Modular Exponentiation

x	1	2	3	4	5	6
1	1	2	3	4	5	6
2	2	4	6	1	3	5
3	3	6	2	5	1	4
4	4	1	5	2	6	3
5	5	3	1	6	4	2
6	6	5	4	3	2	1

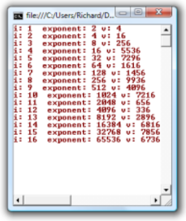
a	a^1	a^2	a^3	a^4	a^5	a^6
1	1	1	1	1	1	1
2	2	4	1	2	4	1
3	3	2	6	4	5	1
4	4	2	1	4	2	1
5	5	4	6	2	3	1
6	6	1	6	1	6	1

Arithmetic mod 7

Fast exponentiation
Repeated Squaring

```
namespace _311ConsoleApp {
    class Program {
        static void Main(string[] args) {
            FastExp(2, 16, 10000);
            System.Console.ReadLine();
        }

        static int FastExp(int x, int n, int modulus) {
            long v = (long)x;
            int exp = 1;
            for (int i = 1; i <= n; i++) {
                v = (v * v) % modulus;
                exp = exp + exp;
                System.Console.WriteLine("i: " + i
                    + "  exponent: " + exp + " v: " + v);
            }
            return (int)v;
        }
    }
}
```



Primality

An integer *p* greater than 1 is called *prime* if the only positive factors of *p* are 1 and *p*.

A positive integer that is greater than 1 and is not prime is called composite.

Fundamental Theorem of Arithmetic: Every positive integer greater than 1 has a unique prime factorization

GCD, LCM and Factoring

$a = 2^3 \bullet 3 \bullet 5^2 \bullet 7 \bullet 11 = 46{,}200$

$b = 2 \bullet 3^2 \bullet 5^3 \bullet 7 \bullet 13 = 204{,}750$

$GCD(a, b) = 2^{\min(3,1)} \bullet 3^{\min(1,2)} \bullet 5^{\min(2,3)} \bullet 7^{\min(1,1)} \bullet 11^{\min(1,0)} \bullet 13^{\min(0,1)}$

$LCM(a, b) = 2^{\max(3,1)} \bullet 3^{\max(1,2)} \bullet 5^{\max(2,3)} \bullet 7^{\max(1,1)} \bullet 11^{\max(1,0)} \bullet 13^{\max(0,1)}$

Euclid's Algorithm

- GCD(x, y) = GCD(y, x mod y)

```
int GCD(int a, int b){   /* a >= b,   b > 0 */
      int tmp;
      int x = a;
      int y = b;
      while (y > 0){
            tmp = x % y;
            x = y;
            y = tmp;
      }
      return x;
}
```


Multiplicative Inverse mod m

Suppose GCD(a, m) = 1

By Bézoit's Theorem, there exist integers s and t such that sa + tm = 1.

s is the multiplicative inverse of a:

$$1 = (sa + tm) \bmod m = sa \bmod m$$

Induction proofs

P(0)
$\forall$ k (P(k) $\rightarrow$ P(k+1))
$\therefore$ $\forall$ n P(n)

1. Prove P(0)
2. Let k be an arbitrary integer ≥ 0
   3. Assume that P(k) is true
   4. ...
   5. Prove P(k+1) is true
6. P(k) $\rightarrow$ P(k+1) — Direct Proof Rule
7. $\forall$ k (P(k) $\rightarrow$ P(k+1)) — Intro $\forall$ from 2-6
8. $\forall$ n P(n) — Induction Rule 1&7

Strong Induction

P(0)
$\forall$ k ((P(0) $\wedge$ P(1) $\wedge$ P(2) $\wedge$ … $\wedge$ P(k)) $\rightarrow$ P(k+1))
$\therefore$ $\forall$ n P(n)

Recursive definitions of functions

- F(0) = 0; F(n + 1) = F(n) + 1;
- G(0) = 1; G(n + 1) = 2 $\times$ G(n);
- 0! = 1; (n+1)! = (n+1) $\times$ n!
- $f_0 = 0$; $f_1 = 1$; $f_n = f_{n-1} + f_{n-2}$

Strings

- The set Σ^* of strings over the alphabet Σ is defined
  - Basis: $\lambda \in S$ (λ is the empty string)
  - Recursive: if $w \in \Sigma^*$, $x \in \Sigma$, then $wx \in \Sigma^*$
- Palindromes: strings that are the same backwards and forwards.
  - Basis: λ is a palindrome and any $a \in \Sigma$ is a palindrome
  - If *p* is a palindrome then a*p*a is a palindrome for every $a \in \Sigma$

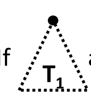
Function definitions on recursively defined sets

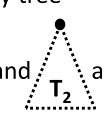
Len(λ) = 0;
Len(wx) = 1 + Len(w); for $w \in \Sigma^*$, $x \in \Sigma$

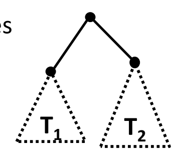
Concat(w, λ) = w for $w \in \Sigma^*$
Concat(w_1,w_2x) = Concat(w_1,w_2)x for w_1, w_2 in Σ^*, $x \in \Sigma$

Prove:
Len(Concat(x,y))=Len(x)+Len(y) for all strings x and y

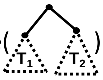
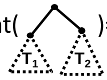
Rooted Binary trees

- Basis: ● is a rooted binary tree
- Recursive Step: If 

and 

are rooted binary trees then so is: 

Functions defined on rooted binary trees

- size(●)=1
- size() = 1+size(T_1)+size(T_2)
- height(●)=0
- height()=1+max{height(T_1),height(T_2)}

Prove:
For every rooted binary tree T, size(T) $\le 2^{\text{height(T)+1}}$ -1

Regular Expressions over Σ

- Each is a "pattern" that specifies a set of strings
- Basis:
  - $\varnothing$, λ are regular expressions
  - ***a*** is a regular expression for any $a \in \Sigma$
- Recursive step:
  - If **A** and **B** are regular expressions then so are:
    - (**A** $\cup$ **B**)
    - (**AB**)
    - **A***

Regular Expressions

- ***0****
- ***0*1****
- (***0*** $\cup$ ***1***)*
- (***0*1****)*
- (***0*** $\cup$ ***1***)* ***0110*** (***0*** $\cup$ ***1***)*
- (***0*** $\cup$ ***1***)* (***0110*** $\cup$ ***100***)(***0*** $\cup$ ***1***)*

Context-Free Grammars

- Example: **S** $\rightarrow$ 0**S**0 | 1**S**1 | 0 | 1 | λ
- Example: **S** $\rightarrow$ 0**S** | **S**1 | λ

Sample Context-Free Grammars

- Grammar for $\{0^n1^n : n \ge 0\}$ all strings with same # of 0's and 1's with all 0's before 1's.
- Example: **S** $\rightarrow$ **(S)** | **SS** | λ

Building in Precedence in Simple Arithmetic Expressions

- **E** – expression (start symbol)
- **T** – term **F** – factor **I** – identifier **N** - number

E$\rightarrow$ **T** | **E**+**T**
T$\rightarrow$ **F** | **F***T**
F$\rightarrow$ (**E**) | **I** | **N**
I $\rightarrow$ x | y | z
N$\rightarrow$ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

BNF for C

```
statement:
  ((identifier | "case" constant-expression | "default") ":")*
  (expression? ";" |
   block |
   "if" "(" expression ")" statement |
   "if" "(" expression ")" statement "else" statement |
   "switch" "(" expression ")" statement |
   "while" "(" expression ")" statement |
   "do" statement "while" "(" expression ")" ";" |
   "for" "(" expression? ";" expression? ";" expression? ")" statement |
   "goto" identifier ";" |
   "continue" ";" |
   "break" ";" |
   "return" expression? ";"
  )

block: "{" declaration* statement* "}"

expression:
  assignment-expression%

assignment-expression: (
    unary-expression (
      "=" | "*=" | "/=" | "%=" | "+=" | "-=" | "<<=" | ">>=" | "&=" |
      "^=" | "|="
    )
  )* conditional-expression

conditional-expression:
  logical-OR-expression ( "?" expression ":" conditional-expression )?
```

49

Definition of Relations

Let A and B be sets,
A binary relation from A to B is a subset of $A \times B$

Let A be a set,
A binary relation on A is a subset of $A \times A$

Let R be a relation on A

R is reflexive iff $(a,a) \in R$ for every $a \in A$

R is symmetric iff $(a,b) \in R$ implies $(b, a) \in R$

R is antisymmetric iff $(a,b) \in R$ and $a \neq b$ implies $(b,a) \notin R$

R is transitive iff $(a,b) \in R$ and $(b, c) \in R$ implies $(a, c) \in R$

Combining Relations

Let R be a relation from A to B
Let S be a relation from B to C
The composite of R and S, $S \circ R$ is the relation from A to C defined

$S \circ R = \{(a, c) \mid \exists$ b such that $(a,b) \in R$ and $(b,c) \in S\}$

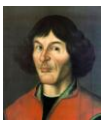
Relations

$(a,b) \in$ Parent: b is a parent of a
$(a,b) \in$ Sister: b is a sister of a
Aunt = Sister $\circ$ Parent
Grandparent = Parent $\circ$ Parent

$R^2 = R \circ R = \{(a, c) \mid \exists$ b such that $(a,b) \in R$ and $(b,c) \in R\}$

$R^0 = \{(a,a) \mid a \in A\}$
$R^1 = R$
$R^{n+1} = R^n \circ R$

$S \circ R = \{(a, c) \mid \exists$ b such that $(a,b) \in R$ and $(b,c) \in S\}$

(Anderson, Copernicus) $\in$ Advisor^{23}


(Beame, Galileo) $\in$ Advisor^{17}

n-ary relations

Let $A_1, A_2, \ldots, A_n$ be sets. An n-ary relation on these sets is a subset of $A_1 \times A_2 \times \ldots \times A_n$.

Student_ID	Name	GPA
328012098	Knuth	4.00
481080220	Von Neuman	3.78
238082388	Russell	3.85
238001920	Einstein	2.11
1727017	Newton	3.61
348882811	Karp	3.98
2921938	Bernoulli	3.21
2921939	Bernoulli	3.54

Student_ID	Major
328012098	CS
481080220	CS
481080220	Mathematics
238082388	Philosophy
238001920	Physics
1727017	Mathematics
348882811	CS
1727017	Physics
2921938	Mathematics
2921939	Mathematics

Matrix representation for relations

Relation R on A={a_1, … a_p}

$$m_{ij} = \begin{cases} 1 \text{ if } (a_i, a_j) \in R, \\ 0 \text{ if } (a_i, a_j) \notin R. \end{cases}$$

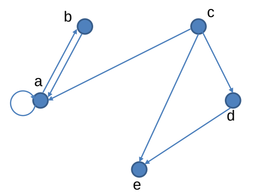
{(1, 1), (1, 2), (1, 4), (2,1), (2,3), (3,2), (3, 3) (4,2) (4,3)}

$$\begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

Representation of relations

Directed Graph Representation (Digraph)

{(a, b), (a, a), (b, a), (c, a), (c, d), (c, e) (d, e) }



Paths in relations

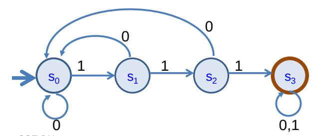
Let R be a relation on a set A. There is a path of length n from a to b if and only if $(a,b) \in R^n$

(a,b) is in the transitive-reflexive closure of R if and only if there is a path from a to b. (Note: by definition, there is a path of length 0 from a to a.)

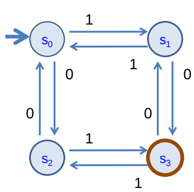
Finite state machines

States
Transitions on inputs
Start state and finals states
The language recognized by a machine is the set of strings that reach a final state

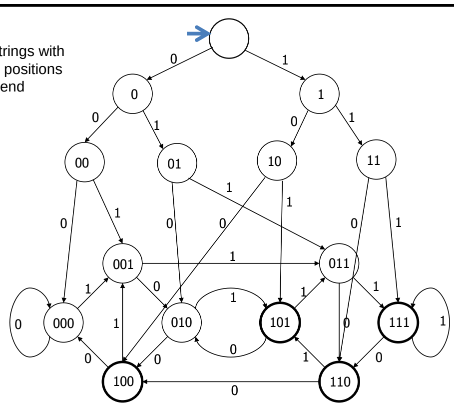
State	0	1
s_0	s_0	s_1
s_1	s_0	s_2
s_2	s_0	s_3
s_3	s_3	s_3



Accepts strings with an odd number of 1's and an odd number of 0's

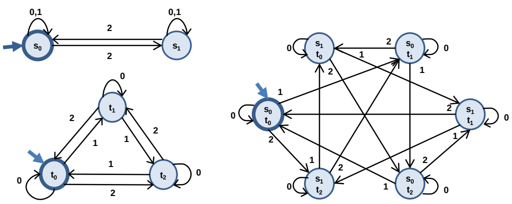


Accept strings with a 1 three positions from the end



Product construction

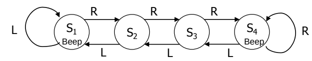
- Combining FSMs to check two properties at once
  - New states record states of both FSMs



State machines with output

	Input		Output
State	L	R	
s_1	s_1	s_2	Beep
s_2	s_1	s_3	
s_3	s_2	s_4	
s_4	s_3	s_4	Beep

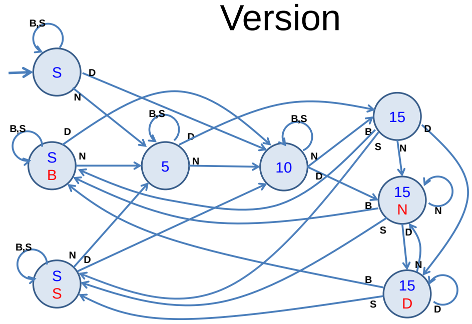
"Tug-of-war"



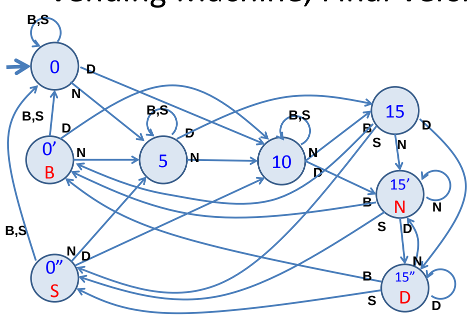
Vending Machine

Enter 15 cents in dimes or nickels
Press S or B for a candy bar

Vending Machine, Buggy Version

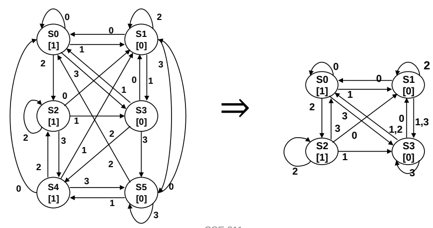


Vending Machine, Final Version



State minimization

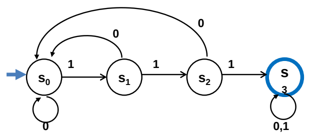
Finite State Machines with output at states



Another way to look at DFAs

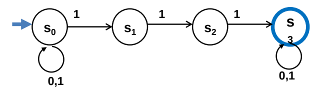
Definition: The label of a path in a DFA is the concatenation of all the labels on its edges in order

Lemma: x is in the language recognized by a DFA iff x labels a path from the start state to some final state

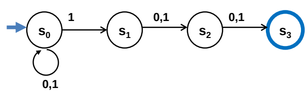


Nondeterministic Finite Automaton (NFA)

- Graph with start state, final states, edges labeled by symbols (like DFA) but
  - Not required to have exactly 1 edge out of each state labeled by each symbol - can have 0 or >1
  - Also can have edges labeled by empty string λ
- Definition: x is in the language recognized by an NFA iff x labels a path from the start state to some final state



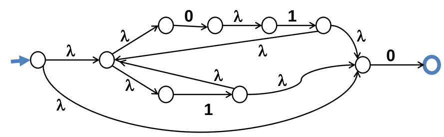
Nondeterministic Finite Automaton



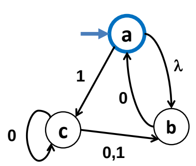
Accepts strings with a 1 three positions from the end of the string

Building a NFA from a regular expression

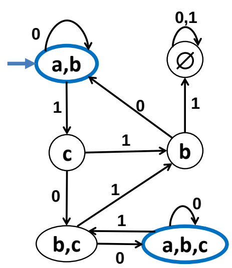
(01 ∪1)*0



NFA to DFA: Subset construction



NFA



DFA

The set B of binary palindromes cannot be recognized by any DFA

Consider the infinite set of strings

S={λ, 0, 00, 000, 0000, ...}

Claim: No two strings in S can end at the same state of any DFA for B, so no such DFA can exist

Proof: Suppose $n \neq m$ and 0^n and 0^m end at the same state p.

Since 0^n10^n is in B, following 10^n after state p must lead to a final state.

But then the DFA would accept 0^m10^n which is a contradiction

Cardinality

- A set S is *countable* iff we can write it as $S=\{s_1, s_2, s_3, \ldots\}$ indexed by $\mathbb{N}$
- Set of integers is countable
  - {0, 1, -1, 2, -2, 3, -3, 4, . . .}
- Set of rationals is countable
  - "dovetailing"

1/1	1/2	1/3	1/4	1/5	1/6	1/7	1/8	…
2/1	2/2	2/3	2/4	2/5	2/6	2/7	2/8	…
3/1	3/2	3/3	3/4	3/5	3/6	3/7	3/8	…
4/1	4/2	4/3	4/4	4/5	4/6	4/7	4/8	…
5/1	5/2	5/3	5/4	5/5	5/6	5/7	…	
6/1	6/2	6/3	6/4	6/5	6/6	…		
7/1	7/2	7/3	7/4	7/5	….			
…	…	…	…	…				

- Σ^* is countable
  - {0,1}* = {0,1,00,01,10,11,000,001,010,011,100,101,...}
- Set of all (Java) programs is countable

The real numbers are not countable

- "diagonalization"

D= 0.

		1	2	3	4	5	6	7	8	9	…
r_1	0.	5^{1}	0	0	0	0	0	0	0	…	…
r_2	0.	3	3^{5}	3	3	3	3	3	3	…	…
r_3	0.	1	4	2^{5}	8	5	7	1	4	…	…
r_4	0.	1	4	1	5^{1}	9	2	6	5	…	…
r_5	0.	1	2	1	2	2^{5}	1	2	2	…	…
r_6	0.	2	5	0	0	0	0^{5}	0	0	…	…
r_7	0.	7	1	8	2	8	1	8^{5}	2	…	…
r_8	0.	6	1	8	0	3	3	9	4^{5}	…	…
…	….	…	….	….	…	…	…	…	…	……	

General models of computation

Control structures with infinite storage
Many models
- Turing machines
- Functional
- Recursion
- Java programs

Church-Turing Thesis
Any reasonable model of computation that includes all possible algorithms is equivalent in power to a Turing machine

What is a Turing Machine?

Halting Problem

- Given: the code of a program **P** and an input **x** for **P**, i.e. given **(<P>,x)**
- Output: **1** if **P** halts on input **x**
  0 if **P** does not halt on input **x**

Theorem (Turing): There is no program that solves the halting problem
"The halting problem is undecidable"

Suppose H(<p>,x) solves the Halting problem

Does **D** halt on input **<D>**?

Function **D(x)**:
if **H(x,x)=1** then
  while (true); /* loop forever */
else
  no-op; /* do nothing and halt */
endif

D halts on input **<D>**

⇔ **H** outputs **1** on input **(<D>,<D>)**
 [since **H** solves the halting problem and so **H(<D>,x)** outputs **1** iff **D** halts on input **x**]

⇔ **D** runs forever on input **<D>**
 [since **D** goes into an infinite loop on **x** iff **H(x,x)=1**]

Does a program have a divide by 0 error?

Input: A program <**P**> and an input string **x**

Output: **1** if **P** has a divide by 0 error on input **x**

0 otherwise

Claim: The divide by zero problem is undecidable

Program equivalence

Input: the codes of two programs, <**P**> and <**Q**>

Output: **1** if **P** produces the same output as **Q** does on every input

0 otherwise

Claim: The equivalent program problem is undecidable

That's all folks!

Teaching evaluation

- Please answer the questions on both sides of the form. This includes the ABET questions on the back